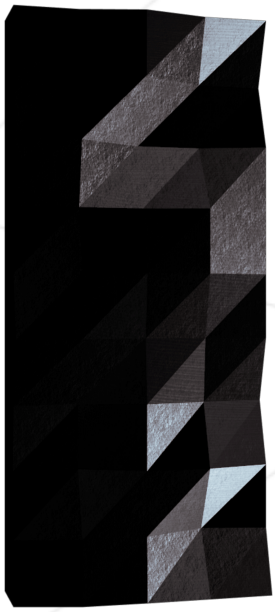

PISCINE OBJECT

Module 01 - Relationship

Summary: This document will introduce you to another important part of Object Oriented Programming : the object propriety.

Version: 1.00

Contents

Chapter I

Preamble

In Object Oriented Programming, objects can have relationships with one another, allowing them to interact and share data. The way objects are related to one another can be categorized into different types based on the degree of their dependence and ownership.

For example, there is a relationship where one object is made up of other objects, with the contained objects being necessary for the functionality of the main object. This relationship represents the strong bond between objects and the reliance of one object on the other.

Another type of relationship exists where objects have a loose connection, and one object can function independently without the other. This relationship represents the loose coupling between objects.

Finally, there is a relationship where objects have a common responsibility, but neither object is dependent on the other. This relationship represents a collaboration between objects.

Chapter II

Introduction

In the following example, we will use a computer system as an object to illustrate different relationships in object-oriented programming. By using a computer system as an object, we can see how these relationships can be applied to real-world situations and help to organize, maintain, and extend code.

The central processing unit (CPU) and the screen are objects that can have a loose relationship with one another, meaning that the screen is a separate object but can be associated with the computer and contribute to its functionality. The screen can exist independently, but when associated with the computer it helps to enhance its display capabilities.

On the other hand, the CPU and the graphics processing unit (GPU) can have a strong relationship, where the GPU is part of the computer and cannot exist without it. The computer can be seen as the main object and the GPU as a contained object, necessary for the computer's functionality.

In addition, the computer and the printer objects can have an relationship, where the computer and the printer have a common responsibility of printing documents, but neither object is dependent on the other. The computer can print to any printer that is connected, and the printer can receive print jobs from any computer. The two objects collaborate to perform the task of printing, but they can function independently as well.

Finaly, the central processing unit is the parent class and the processor object is the child class. The processor object inherits the properties and behaviors of the CPU class, such as the clock speed and number of cores. However, the processor object also specializes the properties and behaviors of the CPU by adding specific information such as the make and model of the processor.

You will find in this module 4 exercices, related to this subject !
Try to guess what name correspond to what kind of relationship !

Chapter III

General rules

- Your program should not crash in any circumstances (even when it runs out of memory), and should not quit unexpectedly.
  If it happens, your project will be considered non-functional and your grade will be `0`.
- You have to turn in a `Makefile` which will compile your source files. It must not relink.
- Your `Makefile` must at least contain the rules:
  `$(NAME)`, `all`, `clean`, `fclean` and `re`.
- Compile your code with `c++` and the flags `-Wall -Wextra -Werror`
- Your code must comply with the **C++ 98 standard**. Then, it should still compile if you add the flag `-std=c++98`.
- Try to always develop using the most `C++` features you can (for example, choose `<cstring>` over `<string.h>`). You are allowed to use `C` functions, but always prefer their `C++` versions if possible.
- Any external library and `Boost` libraries are forbidden.

Chapter IV

Exercice 00 : Putting it into practice

	Exercise 00
Exercice 00 : Putting it into practice	
Turn-in directory : *ex*00/	
Files to turn in : `ex00/*.cpp, ex00/*.hpp, ex00/main.cpp, Makefile`	
Allowed functions : `standard STD containers and structure`	

We can only strongly recommend that you read the preamble and the introduction, you may find some really useful knowledge there.

During the evaluation, you will be asked questions about specific terms in Object Oriented programming, so feel free to search on the internet for definition, and try to made them yours !

As we're here to evaluate architecture and code planning, you're allowed to place method's code directly inside header files, but feel free to create .cpp if you think it's clearer to separate it from declaration.

Note that we expect you to place some output in pretty much every function, method, constructor, operator and destructor you will create in this module.

Those output will be used during evaluation to locate where your code goes, and see what it does, and in what order.

So place some output everywhere you find it necessary !

IV.1 Composition

In this section, you must create the following structures:

- A structure `Position`, having a *int* x, a *int* y, and a *int* z.
- A structure `Statistic`, having two attributes: *int level* and *int exp*.
- A class `Worker`, which must contain a *Position coordonnee* and a *Statistic stat*.

The worker must have the Position and Statistic as composition.

`Be ready to be asked during the evaluation about what does this sentence mean`

IV.2 Aggregation

You must add a class `Shovel` with a method `use`. The Shovel must be able to be given to a `Worker` and then taken away if necessary.
In case of deletion of the `Worker`, the `Shovel` must not be destroyed.

You must perform the following tasks:

- Create a structure `Shovel`, with an attribute `numberOfUses`.
- Modify the class `Worker` to add a `Shovel` object, in a way that allows it to be given and taken away.
- Ensure that the `Shovel` is not deleted in case of deletion of the `Worker` object.
- Ensure that, if the tool is already given to a worker, giving it to another worker does remove it from the first

IV.3 Inherence

You must add an abstract class `Tool` with a pure virtual method `use`. The `Shovel` must inherit from `Tool`. You must also create another class `Hammer`, which also inherits from `Tool`.
Each tool must have a number of uses and a `use` method that indicates what tool it is.
The `Worker` must now be able to have multiple tools simultaneously and be able to return them.

You must perform the following tasks:

- Create an abstract class `Tool`.
- Make the class `Shovel` inherit from `Tool`.
- Create a class `Hammer` that inherits from `Tool`.
- Ensure that tools can be given to a `Worker` and taken away.
- Ensure that, if the tool is already given to a worker, giving it to another worker does remove it from the first

IV.4 Association

You must create a Workshop class that has a list of Workers who can work in the Workshop. The Workshop class must have a way for Workers to sign up for the workshop's worker list and be able to leave it freely.

You are tasked with the following:

1. Create a class Workshop
2. Allow the Workshop class to register Workers and release them if requested
3. Allow Workers to register at multiple Workshops simultaneously
4. The Worker class should have a method "work", to execute work if he's registred to a workshop.
5. The Workshop should have a method "executeWorkDay", launching the day for every Worker registred inside it.

Chapter V

Bonuses

We propose the following bonuses :

- Create a GetTool<ToolType>() method in the Worker class to get the first tool of the right type, or nullptr otherwise.
- Make sure that each workshop only accepts workers with a specific tool. If a worker registers but does not have the necessary tool, he should not join the workshop.
- Also, if a worker loses his tool, he should be automatically released from the workshop he's working on.

The bonus part will only be assessed if the mandatory part is PERFECT. Perfect means the mandatory part has been integrally done and works without malfunctioning. If you have not passed ALL the mandatory requirements, your bonus part will not be evaluated at all.

Chapter VI

Submission and peer-evaluation

For this evaluation, there will be verification that you have correctly encapsulate the classes, and there will also be verification about what kind of relationship you have created between structures. There will be also questions about general definition, to check if you have understund the basics principles.

The idea is to force you to THINK of what you do, and not simply follow rules without fully understanding why there are here.

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.